一种 XML 解析器技术的研究与实现

曹风华
(内蒙古财经学院 计算机信息管理学院,内蒙古 呼和浩特 010070)

摘 要：介绍了 XML 解析的详细过程，设计并实现了一个特定 Schema 的 XML 解析器的自动生成工具。该生成工具以一个 XML Schema 文件作为输入,输出一个 JavaCC 词法和语法规格说明文件,然后在 JavaCC 工具的帮助下,生成一个基于特定 XML Schema 的 XML 解析器。实验证明,这个生成解析器能够对 XML 文档进行解析的同时,验证其有效性。

关键词：XML 解析器;基于特定模式;验证;解析器生成器;JavaCC
中图分类号：TP312 **文献标识码**：A **文章编号**：1674-7720(2011)21-0006-05

Research and implementation of an XML parser technology

Cao Fenghua
(Computer Information Manage College, Inner Mongolia Finance and Economics College, Hohhot 010070, China)

Abstract: This paper introduces the analytical process, the detailed XML design and implementation of a particular Schema XML parser automatic generation tool. This generation tool to an XML Schema files as input, output a JavaCC lexical and grammatical specification file, and then in the JavaCC tools help, generates a based on particular XML Schema XML parser, the experiment proved, can be in parser XML document at the same time, to verify the validity.

Key words: XML parser; based on the specific pattern; validation; parser generator; JavaCC

XML[1](Extensible Markup Language)是一种可扩展标记语言,可以用来定义其他的标记语言。自从 XML 成为 W3C 推荐标准以来,XML 以其简单、可扩展性、自描述性、平台中立的特点,正迅速成为 Web 上信息表示与数据交换的标准[2]。目前众多国际著名公司都宣称其产品中支持 XML，促使 XML 成为下一代 Web 的发展方向。越来越多的网站和 Web 的应用使用 XML 技术进行信息发布和数据交换,XML 已成为一种备受瞩目的技术,甚至被誉为互联网上的世界语。XML 现已被广泛应用在各种领域,如电子商务、企业协作、Web 服务等。XML 解析器是 XML 应用的基础,XML 本身只是以纯文本对数据进行编码的一种格式,要想利用 XML,或是利用 XML 文件中所编码的数据,必须先将数据从纯文本中解析出来。因此,要求必须有一个能够识别 XML 文档信息的文本文件阅读器(即 XML 解析器),用来解析 XML 文档并提取其中的内容。为了提高数据的正确性和提高系统的可靠性,XML 解析器还要检查 XML 实例文档是否符合模式的定义和约束，这个过程称为 XML 文档的有效性验证[3]。但带有验证功能的解析器通常效率比较低[4]。近年来,有很多旨在提高 XML 的解析和基于 Schema 验证性能的研究，本文在详细分析了 XML 解析器的解析过程的基础上，设计并实现了一个特定 Schema 解析器的生成工具。生成器根据特定的 Schema,自动产生一个递归下降的 XML 解析器。这个生成解析器能够对 XML 文档同时进行解析和有效性验证。

1 XML 与 Schema 简介

可扩展标记语言 XML 是由 World Wide Web Consortium (W3C) 于 1998 年 2 月发布的一种基于文本的数据描述语言的通行标准,与 HTML 类似,XML 是一种标记语言,两者在语法上有密切的联系。不同的是,HTML 着重于如何显示数据,而 XML 的设计宗旨是存储和传输数据,着重于如何描述数据。XML 来源于标准通用标记语言 SGML(Standard Generalized Markup Language),是 SGML 的一个精简子集[5]。XML 有如下的特点：

(1)可扩展性:XML 是一种元标记语言,即 XML 可用来设计和定义标记语言,XML 强大的功能体现在它可以

用来制定自己的标记语言。不同的具体应用领域可以制定专用的标记语言,作为该领域共享数据和交换信息的基础。

(2) 内容与表现分离:XML 使得用户界面和结构数据之间保持独立。XML 描述数据的内容(即数据是什么),而数据呈现方式则通过样式单来表示。内容与表现分离,使相同的数据可以不同的格式在不同的媒体上表现。

(3)结构化:XML 以结构化的方式描述数据。这个特点使得 XML 能够描述复杂的数据结构,同时也为关系数据和层次数据提供一种方便的描述方式。

(4) 可验证:XML 文档的结构和内容由 XML 模式语言(如 DTD,XML Schema 等)定义。利用 XML 文档所对应的 DTD 或 Schema,可以对 XML 文档有效性进行验证,提高了数据的可靠性和可用性。

XML 模式(Schema)指的是一类 XML 文档的结构或是模型,这个模型描述了一个有效 XML 文档内的元素层次结构和允许的内容。模式定义了一个 XML 词汇表,包括元素名称、属性名称等。模式规定了一个 XML 文档允许出现的元素、相应的元素允许出现的属性以及这些元素的层次结构关系。XML 的模式语言有很多,其中包括文档类型定义 DTD (Document Type Definition)、XML Schema、XML 规则语言描述 RELAX (REgular LAnguage description for XML)、XML 树形规则表示 TREX (Tree Regular Expressions for XML)和下一代 RELAX NG(RELAX Next Generation)[6]。

XML Schema 是一种使用 XML 语法的 XML 模式语言。DTD 曾是描述、约束 XML 文档最广泛的方法,但在应用的过程中,DTD 体现出一些局限性。主要表现在语法与 XML 语法不一致,只支持有限的数据类型而不支持命名空间等方面。作为 DTD 的后继者,XML Schema 克服了这些缺陷。XML Schema 区别于 DTD 的主要特性表现在[7]:

(1)XML Schema 本身就是 XML 文档,使得 XML Schema 的处理可以与 XML 一样,一些用来处理 XML 的技术也可以用来处理 XML Schema。

(2)定义了丰富的数据类型,如布尔型、整型、日期时间、URI、十进制数等简单数据类型。

(3)支持用户自定义数据类型。XML Schema 支持从现有的数据类型派生出新的数据类型,类似于面向对象中的继承。

(4)充分支持命名空间。

因此,XML Schema 成为 W3C 的正式推荐标准,并正逐步取代 XML DTD。

2 Schema 解析器生成工具的设计与实现

基于特定 Schema 的 XML 解析器的基本思想是根据某一特定的 Schema,构造一个专用的解析器,这个解析器能够对输入的 XML 文档进行良构检查,同时验证其有效性。基于特定 Schema 解析器将 XML 的解析和验证结合在一起,在一定程度上提高了基于 XML 应用的效率和性能。但这个解析器只适用于由这个 Schema 定义的 XML 实例文档,对于由其他 Schema 定义的 XML 实例文档则无能为力。当 Schema 改变时或者需要另外一个 Schema 定义时,必须重新构造一个解析器。而本文设计并实现了利用 JavaCC 工具自动生成一个特定 Schema 解析器的方法。该方法以一个 Schema 文件为输入,生成一个基于这个 Schema 的解析器。

自动生成特定解析器的基本流程如图 1 所示。由于 Schema 文档本身也是一种 XML 文档,所以完全可以使用通用的 XML 解析器对其解析,也可以构造一个专用于解析 Schema 的解析器,但由于 Schema 的语法比较复杂,构造起来比较困难。一种较容易实现的方法是先将 Schema 转化为 XML 树模型的表示,再转换为 Schema 的抽象模型表示。基于特定 Schema 的 XML 解析器生成工具的基本步骤如下:

(1) 首先利用 JavaCC 构造一个通用的 XML 解析器(GeneralParser)。

(2) 通用 XML 解析器将 Schema 输入文件解析成一个 XML 语法的元素节点树。

(3) 遍历这一 XML 语法的树模型,将其转换为 Schema 语法的抽象模型。

(4)根据 Schema 抽象模型,生成特定解析器的词法和语法规格说明。

(5)利用 JavaCC,生成基于输入 Schema 的专用 XML 解析器。

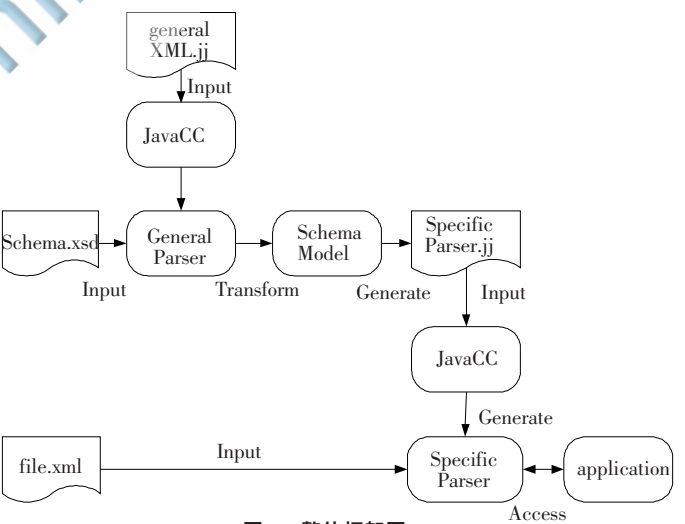

图 1 整体框架图

2.1 构造 XML 解析器

构造 XML 解析器的目的在于解析 Schema 文档,提取其描述和约束 XML 文档结构和内容的信息。XML Schema 遵循 XML 语法,因此可以使用任何通用的 XML 解析器对其解析。下面介绍一个用于解析 XML Schema

文档的 XML 解析器的构造。

由于 XML 文档是可以包含 DTD 声明和 DTD 子集的，所以处理 XML 文档时也应该包含 DTD 的语法的处理。但是 XML Schema 也是一种 XML 文档，一般不会包含 DTD 声明和定义。另外，由 XML Schema 定义的 XML 实例文档通常也不会再用 DTD 定义，所以也不会包含 DTD 的声明或 DTD 子集。因此，在处理 XML Schema 文档时不考虑 DTD 语法的处理；在生成这个 Schema 定义的XML 实例文档的解析器时，也不考虑 DTD 语法的处理。这样有助于简化设计和实现。

2.1.1 构造词法分析器

JavaCC 能根据输入的词法规格说明，产生一个基于 DFA 的词法分析器。因此需要提供一个合适的词法规格说明。

JavaCC 的词法规格说明使用正则表达式定义词法结构，每一个词法记号(Token)名称对应着一个正则表达式，例如：<S：(""|"\t"|"\n"|"\r")+>表示空白空间的词法构成，其中 S 是助记符，而(""|"\t"|"\n"|"\r")+是相应的正则表达式，表示由一个或多个分隔字符组成的字符串，分隔字符包括空格、制表符'\t'、换行符'\n'和'\r'。

XML 规范中表示词法结构的表达式，很多可以比较容易地转换为 JavaCC 词法规格说明的正则表达式形式。但有一些需要特殊的处理才能转换为 JavaCC 可以识别的表示形式。

JavaCC 的词法规格说明由一些词法状态和定义在各个词法状态内的正则表达式组成。生成的词法分析器在分析词法的任何时候都只能处于一个词法状态中。这种机制能够有效地解决多个正则表达式发生冲突的问题。例如，识别标记间的字符数据的正则表达式可以表示为：<CHAR_DATA：(~["<","&","]"]|"]" ~["<","&","]"]|"]"("]")+~["<","&",">"])+("]"|"]*>，这与其他很多记号的正则表达式相冲突，包括标记中的元素名<IDENTIFIER：<NAME>>。这是因为两个正则表达式表示的语言有公共子集，当出现公共子集中的一个串时，词法分析器不知道应该匹配哪一个正则表达式。实际上 JavaCC 只将其匹配为在词法规格文件中较早出现的那个表达式。利用词法状态可以解决这类问题，使元素名只会出现在标记中，而字符数据只出现在标记外，因而可以定义两种词法状态：在标记中的状态和标记外状态，使它们分别在这两个词法状态中识别。词法状态之间的转移可以通过在匹配一个记号后，指定要转移的下一个词法状态来实现。另一种更灵活的方法是在执行词法动作(定义在匹配表达式后执行的 Java 代码)时，调用词法分析器的 SwitchTo()方法，转移到某一指定的状态中。

2.1.2 构造语法分析器

JavaCC 使用自顶向下递归下降的分析方法，并且在需要选择候选式的地方默认向前看一个符号进行判断，因此 JavaCC 使用的也是一种 LL(1)的分析方法。XML 标准中的 EBNF 不是 LL(1)的文法，这样会导致一些选择的冲突，使得语法分析器不能正确地分析语法。虽然 JavaCC 也支持 LL(k)(k>1)的分析方法，即在所有的选择点向前看 k 个符号，但这样会很大程度地降低解析的效率。解决的方法是对 XML 标准中的 EBNF 表示的文法进行改写，使其成为 LL(1)文法。将非 LL(1)文法改写为 LL(1)的过程包括消除左递归和提取左因子。XML 标准中的文法几乎不存在左递归，因此只需要提取左因子。

XML 标准中语法的产生式存在公共左因子的典型例子是元素的产生式。元素及其相关的 EBNF 表达式如表 1 所示。

表 1 元素及其相关的 EBNF 表达式

[39] element	::=	EmptyElemTag | STag content ETag
[44] EmptyElemTag	::=	'<' Name (S Attribute)* S? '/>'
[40] STag	::=	'<' Name (S Attribute)* S? '>'
[41] Attribute	::=	Name Eq AttValue
[42] ETag	::=	'</' Name S? '>'
[43] content	::=	CharData?((element|Reference|CDSect|PI|Comment)CharData?)*

元素产生式的右部是空元素标记或开始标记后跟元素内容和结束标记。其中空元素标记和开始标记有比较长的公共左因子'<'Name(S Attribute)*S?，当语法分析器当前的输入符号为'<'时，无法确定选择 EmptyElemTag 还是 STag content ETag 进行推导，这时解析器就会报错，因此首先要将左因子提取出来。改写元素的产生式为：

element::='<'Name(S Attribute)*S? ('/>'|'>'content ETag) (1)

这样语法分析器遇到'<'时，就不存在选择的问题，当遇到'/>'或'>'时也能确定唯一的子式。但产生式还不是 LL(1)的。其原因在于子式(S Attribute)*S? 也存在选择的冲突。因为 FIRST ((S Attribute)*) ∩FOLLOW ((S Attribute)*)={S}，当语法分析器遇到 S 符号时，不知道应将其匹配为(S Attribute)* 中的 S，还是匹配为后面的 S? 中的 S。因此还需要对产生式进行进一步的改写。

先将子式 AttS::=((S Attribute)*S?)改写为等价的上下文无关文法，然后如图 2 所示提取左因子。

上下文无关文法用 EBNF 表示为：

AttS::=(S(Attribute AttS)?)? (2)

element 改写为：

element::='<'Name AttS('/>'|'>'content ETag) (3)

式(2)中，AttS 是一个非终结符，语法分析的过程中对应于一个过程调用，并且 AttS 中存在着递归，频繁的调用与返回会使系统的消耗较大。因此本文考虑了更好的处理方法：首先对式(2)进析分析，式(2)中存在选择冲突的原因在于(S Attribute)* 的后面存在着 S?，S 表示空白空间，而一般的程序语言的语法分析是不处理空白空

间的，通常在词法分析阶段就把它忽略。是否可以简单地采取忽略空白的做法呢？答案似乎是不可以。因为忽略空白会导致一些良构约束的检查变得困难。例如属性间必须有空白，而且有时 XML 文档中的空白是有意义的，应用程序可能会用到这些空白。但是标记内部的空白是不会提交给应用程序的，为此可以采取灵活的处理方法：由于 XML 的 EBNF 语法中'>'或'/>'前面都必有 S？，因此可以把 S？与'>'或'/>'合并为一个单词。在词法描述中，定义记号 <SRANGLE>::=(<S>)？">"> 和 <SCLOSEDTAG>::=(<S>)？"/>"> 分别代替 <RANGLE>::=">"> 和 <CLOSEDTAG>::="/>">，这样就消除了式(1)中的 S？，而 (S Attribute)* 中的 S 可以保留下来，使得 FIRST((S Attribute)*) ∩ FOLLOW((S Attribute)*)=ϕ，得到元素的产生式为式(4)。式(4)每识别一个属性比式(2)少了一次过程调用，并且消除了递归。

element::='<'Name(S Attribute)*(<SRANGLE>|
<SCLOSEDTAG>content (4)

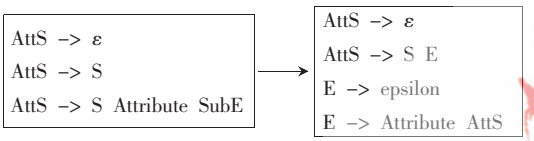

图 2 提取左公告因子

把 XML 标准的 EBNF 文法改写，使得 JavaCC 可以用 LL(1)分析法对其处理。但这种改写有时需要一些技巧，并且不能保证一定有效，改写后的表达式通常比较复杂，也不够直观。一种折中的方法是在少数的一些选择点向前看 2 个符号。

2.1.3 语义分析

XML 中元素间是层次嵌套的关系，因而可以用树来表示元素间的关系，如图 3 所示。树中的节点表示一个 XML 元素，根节点就是 XML。

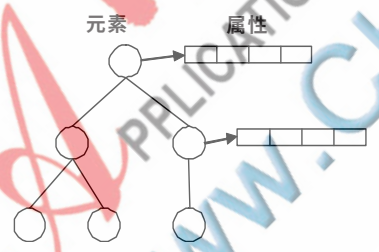

图 3 元素节点树

Schema 文档唯一的根元素(Schema)，各个节点的子节点对应于各个元素的子元素。元素的属性作为节点的属性。图 4 是元素节点及其属性的 UML 类图。

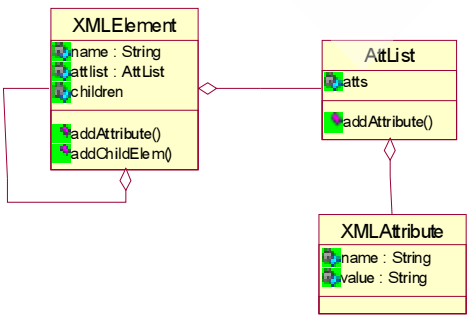

图 4 元素节点类 UML

XML 解析器采用语法制导的翻译方法进行语义分析，即语义分析在语法分析的过程中完成。在 XML 文法产生式的右部中，嵌入相应的语义动作，如创建节点、添加子节点和添加属性等。在分析的过程中，执行所遇到的语义动作。当语法分析结束而没有产生任何错误时，就可以得到一棵包含 Schema 文档元素和属性信息的树。

2.2 模型转换

用本文构造的 XML 解析器分析 Schema 文档，得到一棵以元素为节点的树。这棵树隐含了所需要的描述约束信息，但它是基于 XML 语法的，无法被直接利用，因此必须将其转换为 Schema 语法的对象模型。

2.2.1 XML Schema 抽象模型

为了能够生成一个基于特定 Schema 的解析器，需要一个合适的模型来描述 Schema 文档的结构和各种成分。XML Schema 标准中定义了一个抽象模型。

XML Schema 抽象模型由 13 种模式成分构成，可分为 3 组：主模式成分、二级模式成分、助手模式成分。主模式成分包括简单类型定义、复杂类型定义、属性声明、元素声明；二级模式成分包括属性组定义、本体约束定义、模型组定义、记号声明；助手模式成分包括注解、模型组、粒子、通配符、属性使用。

主模式成分构成模式抽象模型的大部分，其中的声明主要描述了 XML 实例文档的内容，而类型定义尤其是复杂类型定义则描述了 XML 实例文档的结构信息。对 XML 实例文档的验证主要是对 XML 文档的内容和结构两方面的验证。

复杂类型定义使用了其他的模式成分来定义一个元素的内容，这些模式成分包括属性声明、属性组声明、粒子、模型组等，使用属性声明和属性组定义元素的属性，是使用一个粒子类型来定义一个复杂的元素内容模型。复杂类型可以派生新的复杂类型，派生的方式有扩展与约束两种。

粒子(particle)是描述元素内容的一个术语，一个粒子包含两个出现次数约束和一个项。出现次数约束包括最大出现次数和最少出现次数约束。粒子的项可以是一个元素或一个模型组或一个通配符。

一个模型组包含了一组粒子，这些粒子的组合关系是以下三种之一：顺序关系(sequence)、选择关系(choice)、全体关系(all)。顺序关系的粒子必须以顺序的次序出现；选择关系则只出现一个粒子；全体关系的粒子可以任意的次序出现，每一个粒子最多只能出现一次，而且其粒子的项只能是元素。模型组与模型组定义紧密联系而又有所区别，模型组定义是具有名字模式成分，相应的 XML 元素是<group>，而模型组相应的 XML 元素是<all>、<choice>和<sequence>。

XMLSchema 只定义了概念上的抽象模型，具体的实现还需要自己来完成。本文提供了 Schema 抽象模型的

一个简化实现。这个简化的实现暂时忽略了记号声明、本体约束定义、注解和通配符。其他各模式成分也作了相应的简化。简化后实现包含以下各类：

SchemaModel Schema 模型类：包含各种模式成分，如元素声明、属性声明、类型定义、模型组定义等。

XSComponent 抽象的模式成分类：所有模式成分类的父类，实现转换文法接口。

XSElement 元素声明类：表示元素声明模式成分。

XSAttribute 属性声明类：表示属性声明和属性使用模式成分。

XSAttGroup 属性组类：表示属性组定义的模式成分，以及复杂类型定义中所有属性声明的集合。

XSType 抽象类型类：作为简单类型和复杂类型父类。

XSComplexType 复杂类型定义类：表示复杂类型定义模式成分。

XSSimpleType 简单类型定义类：表示简单类型定义模式成分。

XSModelGroup 模型组类：表示模型组及模型组定义模式成分。

XSParticle 粒子类：表示粒子模式成分。

2.2.2 遍历转换

XML 语法的模型转换为 Schema 抽象模型通过遍历元素节点树来完成，遍历使用递归的方式进行。对于元素节点树中的每一个非叶子节点，遍历其子节点，得到相应的子模式成分，然后返回本节点表示的模式成分。

对 XML 的元素节点树的遍历过程实际上也是对 Schema 文档有效性验证的过程。

本文介绍了 XML 解析的详细过程，设计并实现了一个特定 Schema 的 XML 解析器的自动生成工具。这个生成工具以一个 XML Schema 文件作为输入，输出一个 JavaCC 词法和语法规格说明文件，然后在 JavaCC 工具的帮助下，生成一个基于特定 XML Schema 的 XML 解析器。这个解析器能够对 XML 文档进行解释的同时，验证其有效性。

参考文献

[1] 刘芳，肖铁军.XML 应用的基石：XML 解析技术[J].计算机工程与设计，26(19)：2823-2839.2005.

[2] CHIU K，LU W. A Compiler-based approach to schema-specific XML parsing [C]. In：The First International Workshop on High Performance XML Processing，New York，USA，May，2004：17-22.

[3] MATSA M，PERKINS E，HEIFETS A，et al. A high-performance interpretive approach to schema-directed parsing [C]. In：WWW '07：Proceedings of the 16th international conference on World Wide Web，New York，NY，USA，ACM. 2007：1093-1102.

[4] ZHANG W，ENGELEN R V. TDX：a high-performance table-driven XML parser[C]. In：ACM-SE 44：Proceedings of the 44th annual Southeast regional conference，New York，NY，USA，ACM. 2006：726-731.

[5] ZHANG W，ENGELEN R V. A table-driven streaming XML parsing methodology for high-performance Web services [C]. In：ICWS '06：Proceedings of the IEEE International Conference on Web Services (ICWS'06)，Washington，DC，USA，IEEE Computer Society. 2006：197-204.

[6] KOSTOULAS M G，MATSA M，MENDELSOHN N，et al. Xml screamer：an integrated approach to high performance xml parsing，validation and deserialization [C]. In：WWW '06：Proceedings of the 15th international conference on World Wide Web，New York，NY，USA，ACM.2006：93-102.

[7] 左伟明. 即用即查 XML 数据标记语言参考手册 [M].北京：人民邮电出版社，2007.

(收稿日期：2011-07-05)

作者简介：

曹风华，女，1977 年生，研究生，讲师，主要研究方向：Web 数据库系统与应用技术，XML 应用技术。